

Le langage SQL

**La définition des objets de la base de
données**

C. ROELS

I. Les tables

I.1 La création

```
CREATE TABLE nom_table  
( nom_col1 format_col1_default_expr_col1 contrainte_col1 ,  
  nom_col2 format_col2 contrainte_col2 ,  
  ..... ,  
  contraintes_sur_table
```

```
);
```

Unique
Primary key
Foreign key

Not Null
Unique
Primary key
Check

La création d'une table

Des contraintes peuvent être définies au niveau table ou au niveau colonne :

- Une contrainte sur une colonne n'affecte que la colonne
- Une contrainte sur une table peut affecter plusieurs colonnes

Vous pouvez nommer une contrainte.

Si vous ne la nommez pas : un numéro sera assigné afin de la rendre unique dans la base de données.

Les contraintes sur colonnes :

- obliger qu'une colonne ne contienne pas de valeur nulle (**not null**)
- définir qu'une colonne doit contenir une valeur unique dans la table (**unique**)
- définir une colonne en tant que clé primaire (**primary key**)
- indiquer que la valeur d'une colonne doit exister dans une autre table (**foreign key**)
- tester la valeur d'une colonne (**check**)

Les contraintes sur table :

- définir qu'un groupe de colonnes doit contenir une valeur unique dans la table (**unique**)
- définir un groupe de colonnes en tant que clé primaire (**primary key**)
- indiquer que la valeur d'un groupe de colonnes doit exister dans une autre table (**foreign key**)

I. Les tables

I.2 La modification

```
ALTER TABLE nom_table  
    ADD ( col1 format1 ... contr1, col2 format2 ... contr2, ... )  
    ADD CONSTRAINT nom_contrainte ...  
    MODIFY ( col1 format1 , col2 format2, ... )  
    DROP CONSTRAINT nom-contrainte  
    BACKUP ;
```

ALTER TABLE = modification de la structure d'une table

- ajout de colonnes avec ou sans contraintes (ADD)
- ajout de contraintes sur la table (ADD CONSTRAINT)
- modification format de colonne (MODIFY)
- supprimer des contraintes (DROP CONSTRAINT)
- signaler la date de dernière modification de structure (BACKUP)

Si vous avez modifié une table en ajoutant une (des) colonne(s) des vues existantes sur la table (créées avec select *) ne marcheront plus.

Solution :

Supprimer et re-créer les vues.

Autre syntaxe :

Create table as select ... from ... where ... ;

Permet de créer une table à partir d'une autre, tout en récupérant des données de la table initiale.

I. Les tables

I.3 La suppression / renommage

```
DROP TABLE nom_table;
```

Détruit la table, les indexes et les droits créés sur la table.

```
RENAME TABLE nom_table TO nouveau_nom;
```

Renomme la table et
redirige les indexes et les droits créés sur la table.

Dans les 2 cas :

les vues et synonymes ne sont pas détruites,
mais deviennent invalides !

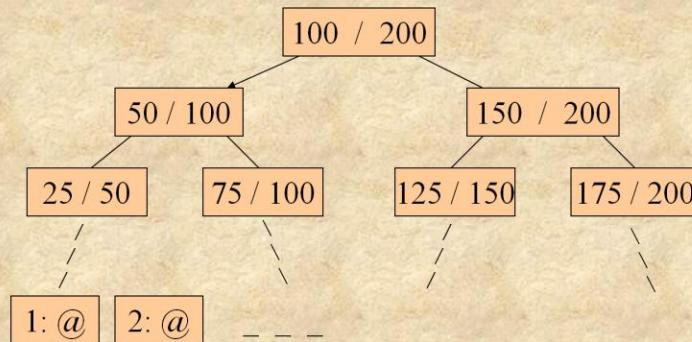
II. Les index

II.1 Définition

Index : accélérateur de recherche

Organisation par défaut : B-tree (balanced tree)

Recherche : dichotomique



index : recherche (sélection) très rapide
ralentit insert , delete, update d'une
colonne indexée

Maximum 16 colonnes dans un index !

Il peut y avoir plusieurs indexes sur une table.

II. Les index

II.2 La création

```
CREATE UNIQUE INDEX nom_index
    ON nom_table ( col1[asc/desc] , col2 [asc/desc] ... )
```

```
CREATE INDEX nom_index
    ON nom_table ( col1[asc/desc] , col2 [asc/desc] ... )
```

Nom_table : indication de la table sur laquelle l'index est créée.

L'index peut être créé sur 1 ou plusieurs colonnes. Par défaut, l'index est trié de façon ascendante. Il est possible d'indiquer un sens de tri pour chaque colonne (asc ou desc).

Exemples :

```
CREATE UNIQUE INDEX i_client_tel ON client (notel) ;
```

```
CREATE INDEX i_cde_cli ON commande(nocli);
```

```
CREATE INDEX i_cli_nom ON client( nomcli, prenomcli );
```

```
CREATE INDEX i_cde_date ON commande(datcde desc);
```

II. Les index

II.2 La suppression

```
DROP UNIQUE INDEX nom_index;
```

```
DROP INDEX nom_index;
```

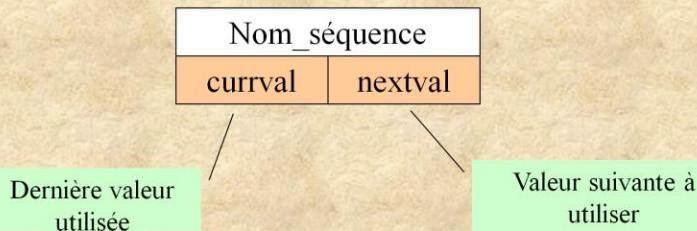
A l'exécution d'une requête, le SGBD vérifie s'il existe des index utilisables sur les tables demandées.

Si aucun index n'est utilisable, la requête sera tout de même exécutée. Elle utilisera une lecture séquentielle (full table scan) en nécessiterait plus de temps que si un index était utilisable (recherche dichotomique si index B-tree).

III. Les séquences

III.1 Définition

Séquence : utilisé afin d'incrémenter la valeur d'une colonne
 (ex. clé primaire)



Dès qu'une séquence a été utilisée :

- le nextval est transféré vers le currvval
- le nextval est incrémenté

REMARQUE :

Une séquence n'est pas rattachée à une table.

Une même séquence pourrait être utilisée lors des insertions dans différentes tables.

Toutefois, afin de garder une certaine logique dans la numérotation, il est préférable de créer une séquence pour chaque table qui nécessite une numérotation automatique.

ACCÈS CONCURRENTS SUR UNE SÉQUENCE

L'incrémentation d'une séquence ne tient pas compte de COMMIT OU ROLLBACK

Ainsi, si deux utilisateurs accèdent à la même séquence, ils utiliseront 2 numéros différents !

III. Les séquences

III.2 La création

```
CREATE SEQUENCE nom-seq
          START WITH      val-init
          INCREMENT BY   val-incr
          MINVALUE       val-min (ou NOMINVALUE)
          MAXVALUE       val-max (ou NOMAXVALUE)
          CYCLE          (ou NOCYCLE);
```

Valeurs minimale et maximale que peut prendre la séquence, par défaut : 1 et 1027

Valeur initiale, 1 par défaut.

Valeur d'incrémentation, 1 par défaut

Cycle : La séquence reprend la valeur val-init quand val-max est atteint.
Nocycle : un message d'erreur sera envoyé quand val-max est atteint.

Exemple :

```
CREATE SEQUENCE seq-client
          INCREMENT BY      10
          START WITH        10
          MINVALUE          10
          MAXVALUE          99 000
          NOCYCLE ;
```

NOTE : la séquence n'est pas liée à la table client !

III. Les séquences

III.3 La modification /suppression

```
ALTER      SEQUENCE nom-seq
            START WITH      val-init
            INCREMENT BY   val-incr
            MINVALUE val-min (ou NOMINVALUE)
            MAXVALUE val-max (ou NOMAXVALUE)
            CYCLE        (ou NOCYCLE) ;

DROP       SEQUENCE nom-seq;
```

Tous les paramètres peuvent être modifiés SAUF la valeur initiale (START WITH).

Exemple :

```
ALTER SEQUENCE seq-client
      MAXVALUE 999 000 ;
```

```
DROP SEQUENCE seq-client;
```

III. Les séquences

III.4 L'utilisation

```
INSERT INTO nom_table (col1, col2, ... )  
VALUES (nom_sequence.NEXTVAL, val2, ... );
```

Une valeur d'une séquence peut être affectée à une colonne NUMERIQUE dans une commande UPDATE et INSERT .

Les pseudo-colonnes doivent être préfixées par le nom de la séquence.

Si la séquence a été créée par un autre utilisateur, il faut également préfixer le nom de la séquence par le nom de l'utilisateur.

Exemples :

Si vous êtes propriétaire de la séquence SEQ_CLI :

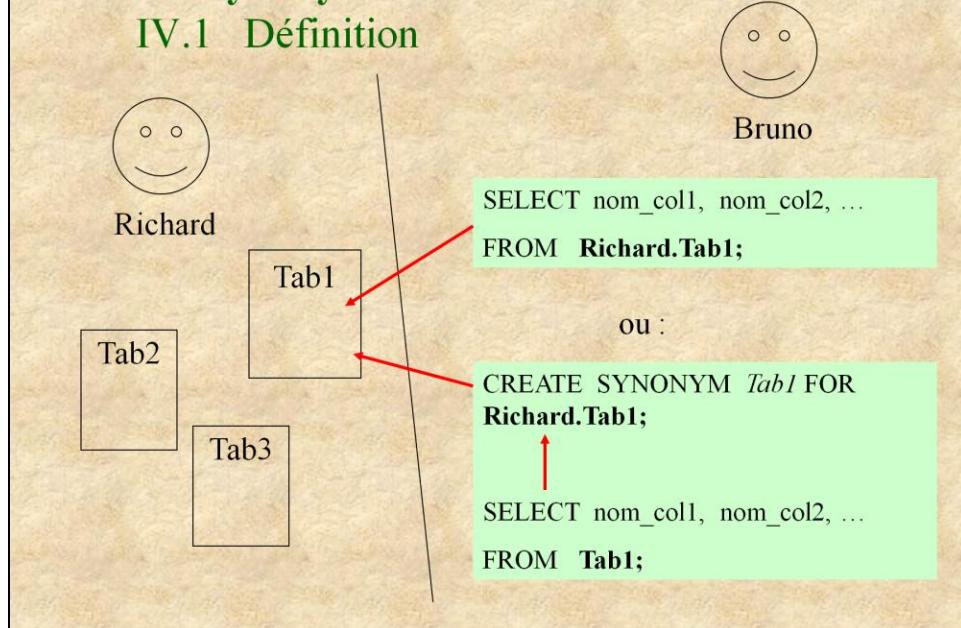
```
Insert into client (nocli, nomcli, pren_cli,...)  
Values (seq_cli.nextval, 'Durand', 'Paul', ...);
```

Si la séquence SEQ_CLI a été créée par l'utilisateur JOHN :

```
Insert into client (nocli, nomcli, pren_cli,...)  
Values (john.seq_cli.nextval, 'Durand', 'Paul', ...);
```

IV. Les synonymes

IV.1 Définition



Lorsque des tables sont créées sous un compte utilisateur, celles-ci sont stockées dans un schéma appartenant à cet utilisateur. Le nom du schéma est identique au compte utilisateur.

Ce mécanisme permet d'isoler les objets appartenant à un utilisateur.

Il est possible d'avoir 2 tables avec un nom identique dans des schémas différents.

Lorsqu'on veut accéder à une table appartenant à un autre utilisateur, il faut :

- que cet utilisateur nous ait donné des droits d'accès sur la table
- qu'on indique le schéma auquel appartient la table, soit directement dans la requête, soit par l'intermédiaire d'un synonyme.

```
SELECT      nom_col1, nom_col2, ...   FROM  Richard.nom_table
```

Afin d'éviter d'avoir à préfixer tous les noms de table ainsi, Bruno peut créer des synonymes :

```
CREATE      SYNONYM  nom_tab1      FOR  Richard.nom_table ;
```

```
SELECT      nom_col1, nom_col2, ...   FROM      nom_tab1;
```

IV. Les synonymes

IV.2 La création

```
CREATE      SYNONYM    nom_synonyme
FOR        [user.] table ;
```

Synonyme valable pour l'utilisateur qui l'a défini

```
CREATE      PUBLIC SYNONYM   nom_synonyme
FOR        [user.] table ;
```

Synonyme utilisable par TOUS les utilisateurs

REMARQUE :

Après création d'un synonyme, vous pouvez référencer la table soit de la façon habituelle, soit par la synonyme.

Exemple :

```
SQL> connect medtra/medtra
SQL> grant select on adherent to carina;
SQL> connect carina/roels
SQL> create synonym adh for medtra.adherent;
```

```
SQL> select count(*) from medtra.adherent;
COUNT(*)
```

46

```
SQL> select count(*) from adh;
COUNT(*)
```

46

IV. Les synonymes

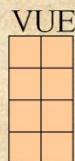
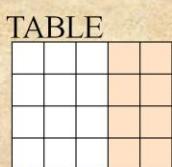
IV.3 La suppression

Il faut être connecté avec le compte ayant crée le synonyme

DROP SYNONYM *nom_synonyme*;

DROP PUBLIC SYNONYM *nom_synonyme*;

V. Les vues



UNE VUE = UNE TABLE VIRTUELLE EXTRAITE d'UNE OU PLUSIEURS TABLES EXISTANTES.

Il est possible d'effectuer des opérations à travers une vue, comme s'il s'agissait d'une table.

Statique

Dynamique

Matérialisée

Une vue statique : contient éventuellement une condition avec une valeur précise.

Une vue dynamique : contient une condition portant sur l'environnement courant.

Une vue matérialisé : récupère et stocke les valeurs obtenus par la vue. Cette technique est utile afin d'optimiser les accès, mais nécessite un rafraîchissement régulier.



FIN

Annexes

CREATE TABLE :

Les types de données

CHAR(n)	type caractère de longueur fixe
VARCHAR(n)	caractère de longueur variable
NUMBER(n)	numérique de type entier
NUMBER(m,n)	numérique avec décimales
DATE	format date interne : année, mois, jour, heure minute, seconde
LONG	texte pouvant contenir 65 535 caractères (1 seul par table)



CREATE TABLE : **La valeur par défaut**

```
Create table client  
(nocli number(2) not null,  
....  
Tauxrem number(2) default 2,  
....  
);
```

Attribution de la valeur
2, à défaut d'une autre
saisie



DEFAULT

définit la valeur par défaut à introduire dans la colonne, si aucune valeur n'est introduite lors d'un insert.

CREATE TABLE : La contrainte NOT NULL, sur colonne

```
Create table client  
(nocli number(2) not null,  
....  
Tauxrem number(2) default 2  
....  
);
```

Rend la saisie obligatoire pour cette colonne



NOT NULL

Une colonne qui fait partie de la clé primaire de la table doit forcément être NOT NULL !

Une colonne qui fait partie d'une clé étrangère n'est pas forcément NOT NULL :

vérifiez la conception de la B.D. afin de déterminer si le lien est obligatoire ou non !

D'autres colonnes peuvent être définies en NOT NULL.

CREATE TABLE : **La contrainte UNIQUE, sur colonne**

Exemple :

```
Create table client  
(nocli number(2) not null,  
....  
Notel char(10) unique,  
....  
);
```

Il sera impossible
d'introduire un numéro
de téléphone identique
pour 2 clients !



UNIQUE

Il est inutile d'indiquer une contrainte UNIQUE sur une colonne qui est définie en tant que clé primaire.

Une clé primaire est par définition unique !

Cette contrainte est donc utilisée pour garantir l'unicité sur d'autres colonnes.

CREATE TABLE :

La contrainte Primary Key, sur colonne

Exemple :

```
Create table client  
(nocli number(2) not null primary key ,  
....  
Notel char(10) unique,  
....  
);
```

La colonne *nocli* doit contenir une valeur unique pour chaque client !



Utilité de la clé primaire :

-lors des insertions dans la table : vérification de l'unicité de la valeur de cette colonne

-Lors des suppressions dans la table : interdiction de suppression s'il existe encore des lignes dans d'autres tables qui font référence à cette clé primaire

CREATE TABLE : **La contrainte CHECK, sur colonne**

Exemple :

```
Create table client  
( nocli number(2) not null primary key ,  
....  
Notel char(10) unique,  
Tauxrem number(2) check( tauxrem in (2,4,6) ) ,  
....  
);
```

La colonne *tauxrem* ne pourra accepter que les valeurs indiquées dans le check



CHECK

Utilisé afin de vérifier la valeur d'une colonne lors d'une insertion ou d'une modification de ligne.

La vérification peut s'exprimer de différentes façons :

colonne IN (valeur1, valeur2, ...)

colonne BETWEEN valeur_num1 AND valeur_num2

colonne = valeur

colonne LIKE 'valeur_partielle%'

etc...

CREATE TABLE :

La contrainte UNIQUE, sur table

Exemple :

Create table client

(nocli number(2) not null,

....

Indtel char(3) ,

Notel char(10) ,

Constraint un_cli_tel unique (Indtel, Notel)

);

Un_cli_tel = nom de la
contrainte

Unique = type de
contrainte

(Indtel, Notel) = colonnes
de la table en cours de
création affectées par la
contrainte



L'unicité est vérifié sur la combinaison des 2 colonnes :

Indtel : Indicatif pays (ex. 033, 032, ...)

Notel : le numéro de téléphone

CREATE TABLE :

La contrainte Primary Key, sur table

Exemple :

```
Create table detail_cde  
(nocde    number(2) not null ,  
codprod  char(5) not null ,  
quantité  number(3) ,  
Constraint pk_detail primary key (nocde, codprod) ,  
....  
);
```

Pk_detail = nom de la contrainte

Primary key = type de contrainte

*(nocde, codprod) = colonnes de la table en cours de création
affectées par la contrainte*



Une contrainte sur colonne peut également être indiquée de cette façon, à la fin de l'instruction CREATE TABLE.

L'utilité :

La contrainte reçoit le nom que l'on indique.

Dans le cas d'une contrainte indiquée derrière la définition d'une colonne, le SGBD attribue lui-même un nom à la contrainte (ex. : SYS000126).

La contrainte peut s'appliquer sur 1 ou plusieurs colonnes.

Le code de création de la table devient plus lisible.

CREATE TABLE :

La contrainte Foreign Key, sur table

Exemple :

```
Create table commande  
(nocde    number(5) not null ,  
 datcde   date       not null ,  
 nocli    number(2) not null ,  
 Constraint pk_cde primary key (nocde),  
 Constraint fk_cde_cli foreign key (nocli)  
           references client(nocli)  
);
```

*Fk_cde_cli = nom de contrainte
Foreign key = type de contrainte
(nocli) = colonne de la table en cours de création affectée par la contrainte*

La colonne nocli de la table commande fait référence à la colonne nocli de la table client.



FOREIGN KEY

- Doit référencer une clé primaire dans une autre table.
- Doit référencer une table , pas une vue !
- Implique que vous êtes propriétaire de la table référencée ou que vous avez le droit de la référencer .
- Rejette tout INSERT ou UPDATE avec une valeur qui n'existe pas dans la table référencée.
- Rejette tout DELETE dans une table si cela invalide une référence dans une autre table.
Sauf si la clé étrangère a été définie avec l'option ON DELETE CASCADE : en cas de suppression dans la table référencée, les enregistrements fils sont supprimés en cascade.

Autre exemple :

```
Create table detail_cde  
(nocde        number(2) not null ,  
 codprod      char(5) not null ,  
 quantité     number(3) ,  
 Constraint pk_detail primary key (nocde, codprod),  
 Constraint fk_detail_cde foreign key (nocde)  
           references commande(nocde) ,  
 Constraint fk_detail_prod foreign key (codprod)  
           references produit(codprod) );
```

ALTER TABLE : **L'ajout d'une colonne**

```
ALTER TABLE client  
ADD noag number(2);
```

Ajout d'une
colonne sans
contrainte

```
ALTER TABLE produit  
ADD date_valide date  
check( date_valide > sysdate);
```

Ajout d'une
colonne avec
contrainte

ADD (col1 format1 ... contr1, col2 format2 ... contr2, ...)



ALTER TABLE : L'ajout d'une contrainte

```
ALTER TABLE client  
ADD CONSTRAINT fk_client_ag foreign key (noag)  
references AGENCE(noag);
```

```
ALTER TABLE client  
ADD CONSTRAINT ck_tauxrem  
CHECK (tauxrem < 10);
```



ALTER TABLE :

La modification d'une colonne

```
ALTER TABLE client  
MODIFY tauxrem number(4,2);
```

Il est possible de modifier le format d'une colonne
tant que la table ne comporte pas de données.



ALTER TABLE :

La suppression d'une contrainte

```
ALTER TABLE client  
DROP CONSTRAINT ck_tauxrem;
```

Il suffit de connaître le nom de la contrainte à supprimer.

Dans le cas où la contrainte a été nommée par le SGBD, il faudra d'abord rechercher le nom de la contrainte dans le dictionnaire des données.



DROP CONSTRAINT nom-contrainte

ALTER TABLE : BACKUP

Indique la date et
l'heure du ALTER
TABLE dans le D.D.

```
ALTER TABLE client  
ADD noag number(2) BACKUP;
```

```
ALTER TABLE client  
ADD CONSTRAINT ck_tauxrem  
CHECK (tauxrem < 10) BACKUP;
```

```
ALTER TABLE client  
MODIFY tauxrem number(4,2) BACKUP;
```

```
ALTER TABLE client  
DROP CONSTRAINT ck_tauxrem BACKUP;
```



Exemples de création de vues statiques

```
Create view V_CDE_C1  
As select * from commande where cial_no = 1;
```

```
Create view V_CA_C1( cial_no, ca )  
As select cial_no, sum( qte * prixprod )  
From commande c, detail_cde d, produit p  
Where c.nocde = d.nocde  
And d.codprod = p.codprod  
And cial_no = 1  
Group by to_char(sysdate, 'mm/yyyy') ;
```



Une vue créée à partir d'une seule table :

Il est possible d'insérer des lignes à travers la vue; y compris des lignes qui ne respectent pas la condition de la vue.

Afin de forcer le contrôle sur la restriction de la vue, il faut ajouter **WITH CHECK OPTION.**

Une vue créée à partir de plusieurs tables :

Il est possible d'effectuer des recherches par l'intermédiaire de la vue.
Les autres opérations ne sont pas possibles.

RESTRICTIONS

- Si la colonne d'une VUE provient d'une fonction (AVG, SUM, MIN, ...),
on ne peut plus la référencer dans la clause WHERE d'un SELECT sur la vue.

Dans le cas où une colonne de la vue est obtenue par une fonction ou par un calcul,

il est utile de renommer la colonne.

- Une vue définie avec un SELECT utilisant une clause GROUP BY ne peut être utilisée pour une jointure.

Exemple de création d'une vue dynamique

```
Create view V_CDE_CIAL  
As      select .....  
From    commande c, commercial ci  
Where   c.cial_no    = ci.cial_no  
And     ci.nom      =  
          (select username from user_users);
```

Sous requête permettant d'obtenir le
compte de l'utilisateur connecté.



Suppression d'une vue dynamique :

```
DROP VIEW V_CDE_CIAL;
```

Exemples de création de vues matérialisées

Types de vues matérialisées :

VM ‘simple’ ou ‘sur clé primaire’

VM ‘sur rowid’

Méthodes de rafraîchissement :

COMPLETE

FAST REFRESH

FORCE

Modes (périodicité) de rafraîchissement :

synchrone, sur commit

asynchrone, à la demande

asynchrone, cyclique



VM ‘simple’ ou ‘sur clé primaire’

Create **MATERIALIZED** view **VM_CDE_S1**

```
As      select nocde, datcde, ...
From    commande c
Where   c.cial_no = 1;
```



VM ‘sur rowid’

Create **MATERIALIZED** view **VM_CDE_S1**

REFRESH with ROWID

```
As      select .....  
From   commande c  
Where  c.cial_no = 1;
```



VM avec COMPLETE REFRESH

Create **MATERIALIZED** view **VM_CDE_S1**

REFRESH COMPLETE

As select;

From commande c

Where c.cial_no = 1;



VM avec FAST REFRESH

Create **MATERIALIZED** view **VM_CDE_S1**

REFRESH FAST

As select

From commande c

Where c.cial_no = 1;

Préalablement:

create materialized view log on commande;



VM avec rafraîchissement synchrone, sur commit

Create **MATERIALIZED** view **VM_CDE**

REFRESCH COMPLETE

As select * From commande c;

Dès que les modifications de la table principale sont validées (commit), elles sont automatiquement répercutées dans la VM.



VM avec rafraîchissement asynchrone, à la demande

Création d'un groupe de rafraîchissement avec ajout d'une VM :

```
execute DBMS_REFRESH.MAKE  
('ref_group_1','MV_CDE',null,null);
```

Exécution du rafraîchissement du groupe :

```
execute DBMS_REFRESH.REFRESH  
('ref_group_1');
```



VM avec rafraîchissement asynchrone, cyclique

Création d'une VM avec rafraîchissement automatique journalier :

```
create materialized view MV_CDE
refresh next sysdate + 1
as select * from COMMANDE;
```

Modification de la périodicité de rafraîchissement :

```
alter materialized view MV_CDE
refresh complete
start with sysdate next sysdate + 2;
```

